AIR FORCE INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY
UNITED STATES AIR FORCE

SCHOOL OF ENGINEERING

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

COMPARISON OF RELIABILITY ALLOCATION OPTIMIZATION METHODS

Thesis

William Z. Spivey
GRE/EE/69-2 2d Lt. USAF

COMPARISON OF RELIABILITY ALLOCATION OPTIMIZATION METHODS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

William Z. Spivey, B.S.M.E.

Second Lieutenant USAF

Graduate Reliability Engineering

December 1969

PREFACE

This study is the result of a suggestion by Professor T. L. Regulinski, who later became my thesis advisor, that the methods of reliability allocation needed further development.

While I was searching through the literature dealing with reliability allocation, I discovered that allocation models had been developed in each of the areas in which I was interested. This reduced my problem from one of developing and comparing an allocation model in each of the three areas of Lagrange multipliers, mathematical programming, and dynamic programming to one of merely comparing the three allocation models that had been previously developed.

The dynamic programming allocation model was developed by John D. Kettle (Ref 10:249) and the Lagrange multiplier allocation model was developed by Hugh Everett III (Ref 4:399). The algorithm presented here for mathematical programming is based on a heuristic argument that was suggested by Peter J. Kolesar (Ref 11:317).

I wish to express my thanks to Professor Regulinski for the helpful advice and constructive criticism without which I probably would never have completed this study.

Preface (Contd)

I also wish to express thanks to my recent bride for her encouragement during the times when I felt that I was making no progress. Finally, I claim responsibility for any errors that may be found in this work.

William Z. Spivey

Contents

Contents (Contd)

List of Figures

List of Tables

ABSTRACT

The reliability allocation problem has previously been solved by methods other than the classical optimization methods. This thesis brings together and compares three allocation models that are based on optimization methods.

It was found that dynamic programming gives the optimal solution to the allocation problem. The model based on Lagrange multipliers gave a solution that was within 0.5% of the optimal solution given by dynamic programming. The mathematical programming algorithm gave a solution that was within 0.6% of the optimal solution.

GRE/EE/69-2

COMPARISON OF RELIABILITY ALLOCATION OPTIMIZATION METHODS

I. INTRODUCTION

Reliability allocation models that are now in use are based on assignment of reliability goals to the various subsystems to fulfill the requirement that the system failure rate equal the sum of the failure rates of the various subsystems. These methods of allocation are based on the assumptions that times to integrant failure are exponentially distributed. The two models most frequently used are the complexity and the failure rate weighting models.

When the complexity model is used, the most complex subsystem is assigned the highest failure rate. This failure rate will be equal to the ratio of the number of integrants in the subsystem under consideration to the number of integrants in the entire system multiplied times the desired system failure rate. The end result of this model is to assign lower reliability goals to the more complex subsystems and higher reliability goals to the more simple subsystems.

The failure rate weighting model assigns failure rates to the various subsystems based on failure rates that have

been achieved on similar subsystems in the past. Parts that have had low failure rates are assigned reliability goals that are harder to achieve than parts that have had high failure rates.

PROBLEM

The problem is to develop three reliability allocation models, one using Lagrange multipliers, one using mathematical programming, and one using dynamic programming. The allocation will be made in such a way that one of the system parameters will be minimized subject to a constraint placed on the system reliability. These system parameters are such as cost, weight, development time, and test cost.

These three models will be compared and the advantages and disadvantages of each will be discussed.

SCOPE

The study will apply the mathematical methods of Lagrange multipliers, mathematical programming, and dynamic programming to the allocation problem. Any discussion of the theory behind these methods will be limited to that necessary for the application of the method.

The comparison will point out the advantages and disadvantages of each method and will also discuss, for each

method, any changes in assumptions that could be made to make the method more general.

ASSUMPTIONS

The main assumptions are that the operation of any subsystem is independent of the operation of any other subsystem and that the system will operate as long as all of the subsystems are operating.

These assumptions imply that the re' bility of the system will be the product of the reliabilities of all subsystems.

$$RS = \prod_{i=1}^{M} R_i \tag{1}$$

Furthermore, each subsystem is assumed to be composed of N identical components, N-1 of which are redundant. The reliability of each subsystem will be

$$R_i = 1-(1-r_i)^{N_i} \tag{2}$$

The subsystem cost is assumed to be a linear function of the number of components in the subsystem.

$$C_i = N_i c_i \tag{3}$$

STANDARDS

The comparison of the allocation models will be made on the basis of optimality of the solution, flexibility of the algorithm, and simplicity of computation.

Optimality of the solution was chosen as a basis for comparison because the main objective of allocation is to do it in the most optimal manner. If the algorithm does not give a solution that is at least near optimal, then it is of no value.

A flexible algorithm is desirable because an algorithm that can be used for a general reliability allocation problem is better than an algorithm that is limited to more restricting assumptions. In particular, a good algorithm should be able to handle the situation where the cost function is not linear.

Simplicity of computation was chosen as a basis for comparison because a very long algorithm that gives an optimal solution is not necessarily superior to a shorter algorithm that gives a near optimal solution. This is especially true in the situation where the model is only a rough approximation of the actual situation.

APPROACH

The approach to solving the problem will be to first

develop the necessary algorithms. Then these algorithms will be applied to solving an allocation problem for a representative system with component reliabilities and costs that are randomly selected. From the results of this allocation, the limitations of each algorithm will be determined and will be compared with the limitations of the other algorithms. Then the advantages and disadvantages of each model will be discussed.

II. LAGRANGE MULTIPLIERS

As most often used, Lagrange multipliers are introduced in context with differentiable functions. In these cases they are used to produce constrained stationary points.

Another technique involving the use of Lagrange multipliers has been practiced at the Weapons Systems Evaluation Division, Institute for Defense Analysis. The objective of this technique is the maximization of a function with constraints, rather than the location of stationary points (Ref 4:399-417).

THEORY

The basic problem can be described by defining an arbitrary set, S, which is the set of all possible actions. Defined on this set, S is a payoff function, H. There are N real valued functions, $C^k(x)$ (k=1,...,N), defined on S which are called the resource functions. The problem is to find

$$\underset{X \text{ contained in } S}{\text{Max}} \quad H(X) \tag{4}$$

subject to $$C^k(X) \leq C^k \tag{5}$$

for all k, where C^k is the maximum permissible value of this resource function.

This technique is based on the following theorem (Ref 4:401).

Theorem 1.

1. Z^k, k=1,...,N, are nonnegative real numbers.
2. X* contained in S maximizes the function

$$H(X) - \sum_{k=1}^{N} Z^k C^k(X)$$

over all X contained in S.

3. X* maximizes H(X) over all those X contained in S such that $C^k(X^*)$ C^k for all k.

This theorem says, that for any choice Z^k, k=1,...N, that if an unconstrained maximum of the new function

$$H(X) - \sum_{k=1}^{N} Z^k C^k(X) \qquad (6)$$

can be found, then this solution is a solution to that constrained maximization problem with constraints equal to the amount of each resource expended in achieving the unconstrained solution.

According to this theorem, it is possible to select an arbitrary, nonnegative set of Z's, find an unconstrained maximum for equation (6) and this gives the solution to a constrained problem. The only difficulty is that there is no way of knowing in advance which constrained problem will be solved. This diffuculty may be avoided through the use of an iterative procedure that will examine the spectrum of

Z's and isolate the desired constrained solution.

For the general problem this procedure does not guarantee a solution in every case. It does guarantee that if a solution can be found, then it will be optimal.

A subclass of the general problem is one that may be called the cell problem. The cell problem is the problem in which there are a number, M, of independent areas into which the resources may be committed and for which the total payoff is the sum of the payoffs from each independent area.

For this type problem there exists for each cell a strategy set, S_i; a payoff function, H_i defined on S_i; and N resource functions C_i^k defined on S_i. The problem is to find a strategy set, one element for each cell, that maximizes the total payoff subject to constraints on the total resource expended. That is, find:

$$\underset{\substack{\text{all choices of } [X_i] \\ X_i \text{ contained in } S_i}}{\text{Max}} \sum_{i=1}^{M} H_i(X_i) \qquad (7)$$

$$\text{subject to} \quad \sum_{i=1}^{M} C_i^k \quad C^k \qquad \text{for all k.} \qquad (8)$$

This problem may be put in the form of equation (6) by noting that

$$S = \sum_{i=1}^{M} S_i \qquad (9)$$

and that $$H(X) = \sum_{i=1}^{M} H_i(X) \qquad (10)$$

and that $$C^k(X) = \sum_{i=1}^{M} C_i^k(X_i) \quad \text{for all } k. \qquad (11)$$

By applying the theorem to equations (9), (10), and (11) it is seen that finding the maximum of the constrained function (7) is equivalent to finding

$$\underset{X \text{ contained in}}{\text{Max}} \prod_{i=1}^{M} S_i \left[\sum_{i=1}^{M} H_i(X_i)\right] - \sum_{k=1}^{N} z^k \left[\sum_{i=1}^{M} C_i^k(X_i)\right]. \qquad (12)$$

By interchanging the order of summation, the function (12) becomes

$$\underset{X \text{ contained in}}{\text{Max}} \prod_{i=1}^{M} S_i \sum_{i=1}^{M} \left[H_i(X_i) - \sum_{k=1}^{N} z^k C_i^k(X_i)\right] \qquad (13)$$

Since $$S = \sum_{i=1}^{M} S_i \qquad (9)$$

the function (13) is maximized by maximizing

$$H_i(X_i) - \sum_{k=1}^{N} z^k C_i^k(X_i) \qquad (14)$$

independently for each cell.

The theorem guarantees that if the Lagrangian in each cell has been correctly maximized then the result is a global maximum for the entire problem.

APPLICATION

Applying this to the problem of reliability allocation, the system reliability is given by

$$RS = \prod_{i=1}^{M} [1-(1-r_i)^{N_i}] \tag{15}$$

Since the logarithm is a monotonic increasing function, maximizing the logarithm is the same as maximizing the function itself. So the payoff is taken to be the logarithm of the reliability.

$$H = Ln(R) = \sum_{i=1}^{M} Ln[1-(1-r_i)^{N_i}] \tag{16}$$

Now the equation is in the form of the cell problem. To maximize equation (16), it is only necessary to maximize independently for each cell or subsystem the expression

$$H_i(N_i) = Ln[1-(1-r_i)^{N_i}]-Z\ c_iN_i \tag{17}$$

Since the functions, $H_i(N_i)$, are concave, each of them can be maximized analytically by differentiation. The integer maximum for all integers equal to or greater than 1 can be found by checking the integers on each side of the real maximum to see which gives the larger payoff. The result is obtained as

$$\frac{d\,H_i(N_i)}{d\,N_i} = \frac{-(1-r_i)^{N_i}\,\mathrm{Ln}(1-r_i)}{[1-(1-r_i)^{N_i}]} - Z\,c_i \qquad (18)$$

or

$$N_i = \frac{\mathrm{Ln}\,(Z\,c_i) - \mathrm{Ln}\,(Z\,c_i - \mathrm{Ln}\,(1-r_i))}{\mathrm{Ln}\,(1-r_i)} \qquad (19)$$

for real N_i.

This formula is applied for each subsystem, $[N_i]$ and $[N_i + 1]$ are examined to find which integer maximizes, then the reliabilities and costs are summed to produce an optimal solution.

This procedure is repeated for a series of Z's greater than zero. For each Z, the maximum reliability for a given cost is found. The series of reliabilities is then examined to find which solution gives the reliability nearest to but greater than the required system reliability. This solution gives the allocation for minimum cost that gives the required system reliability.

Appendix I contains a listing of a computer program to perform the necessary calculations.

III. MATHEMATICAL PROGRAMMING

Sequential Integer Programming

Mathematical programming is defined as a programming procedure in which the goal is to optimize an objective function subject to one or more constraints.

THEORY AND APPLICATION

Consider the problem of trying to improve the reliability of a system by adding one more redundant component to one subsystem. Assume each subsystem already has N_k components. The increase in reliability of the system by adding one component to the ith subsystem will be

$$E_i = \left[\prod_{k=i} 1-(1-r_k)^{N_k}\right]\left[1-(1-r_i)^{N_i+1}\right] - \prod_{k=1}^{M} 1-(1-r_k)^{N_k}. \qquad (20)$$

If the object is to get the greatest reliability improvement per unit cost, the additional redundant component should be added to the subsystem where the quantity A_i is the largest where A_i is defined as in equation (21).

$$A_i = \frac{E_i}{c_i} \qquad (21)$$

The problem is then expanded to the problem of trying to achieve a system reliability, RS>R. It is intuitively appealing that this can be done in an optimal manner by an iterative scheme composed of an unknown number of steps if at each step the set, A_i,(i=1,...,M), is generated and the one additional component is added to the subsystem where the quantity A_i is the greatest. The iterative scheme continues until a system reliability, $R \geq RS$, is achieved.

For the reliability allocation problem at hand, the initial solution was selected such that each subsystem would have a reliability equal to or greater than the desired system reliability, RS.

Appendix II contains a listing of a computer program to make these calculations.

IV. DYNAMIC PROGRAMMING

Dynamic programming has been developed since 1950. The idea is that starting at the last stage of an allocation process, it is possible to ennumerate all of the alternatives. Thus, it is possible to select the alternative that gives the optimum policy for the last stage. Now, since the optimal policy for the last stage is known, it is possible to find the optimum policy for the next to last stage. Then the process is repeated for each stage working from the last stage toward the first stage.

THEORY

The basic theorem of dynamic programming is "The Principle of Optimality." It states that "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision". A proof of this theorem is given by Bellman and Dreyfus (Ref 2:15).

APPLICATION

The application of this theory to a reliability allocation process is as follows:

(1) Plan a sequence for combining the stages or

subsystems. Continue until all subsystems have been combined.

The plan used for the process discussed here is to combine the first and second subsystems, then join this combination with the third subsystem. In general, the Jth subsystem is joined with the combination of the J-1 previous subsystems.

(2) Determine the minimum number of components for each subsystem that will give the subsystem under consideration a reliability greater than or equal to the required system reliability, RS. This will be the base requirement. The optimal policy will consider only the extra costs and reliabilities above the base requirements.

(3) Prepare the cost-reliability sequence for each subsystem.

(4) Combine the first grouping and select the dominating sequence. The dominating sequence is the optimal policy such that for any given entry, R_i, C_i (i=1),...,M), that entry is the cheapest entry in the group with reliability exceeding R_i-1.

(5) Proceed in this same manner until all of the subsystems have been combined.

The allocation is found from the final entry that gives a reliability greater than or equal to the required

system reliability, RS.

The procedure is demonstrated by the following example. Example: Given a system of 4 subsystems, a required system reliability of .99, and component costs and reliabilities as given in Table I, find the optimum allocation of reliability.

Table I
Component Reliability and Cost for Each Subsystem

Subsystem	Integrant			Base
	Cost	Reliability	Unreliability	Requirements
1	10	.90	.10	2
2	15	.95	.05	2
3	13	.93	.07	2
4	17	.92	.08	2

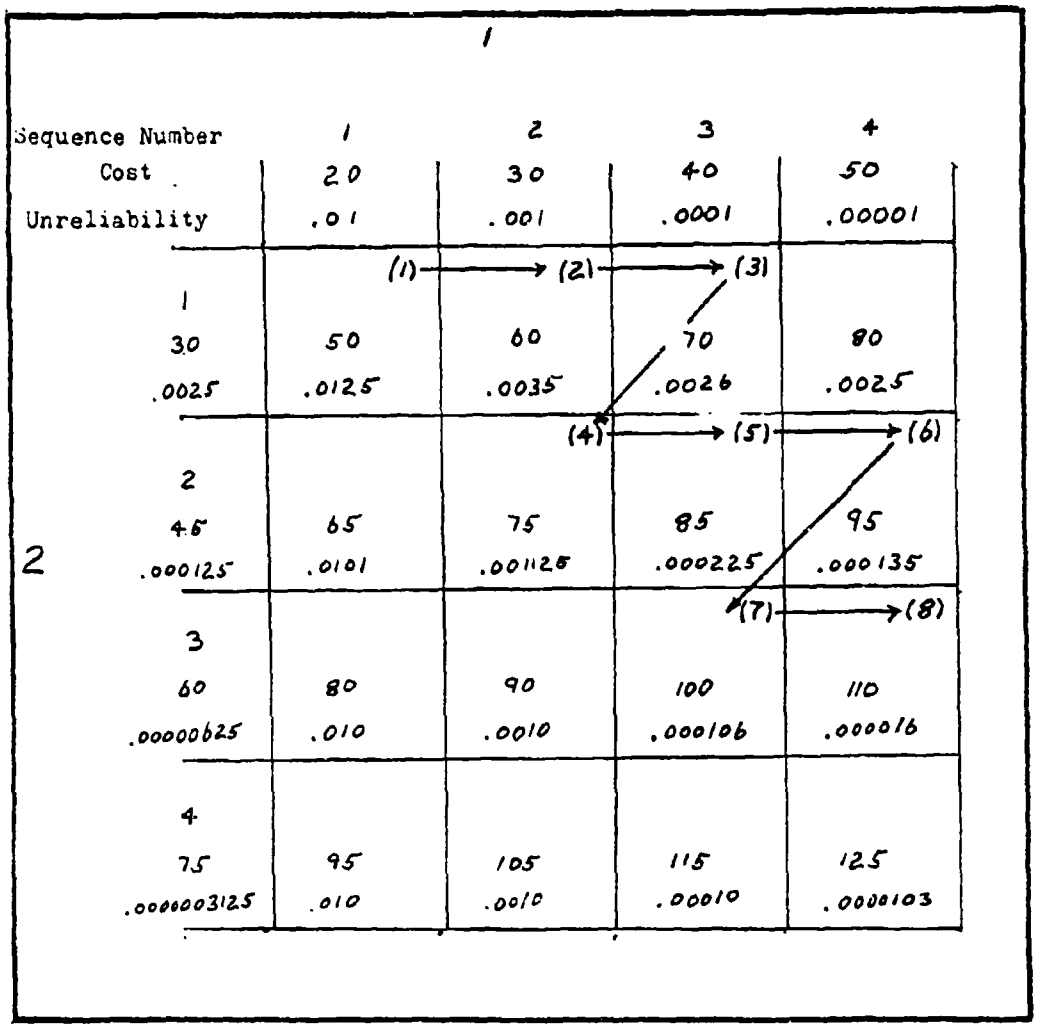

Fig. 1. Combining Subsystems 1 and 2

The order of combination will be 1 and 2, the combination of 1 and 2 with 3, and finally, the combination of 1, 2, and 3 with 4.

The unreliability of the combination of the entries of column i and row j is given by equation (22). When the reliability is close to 1, the unreliability of the

combination is approximately the sum of the unreliability of each. The cost of the combination of the entries in column i and row j is given by equation (23).

		182			
Sequence Number	1	2	3	4	5
Cost	50	60	70	75	85
Unreliability	.0125	.0035	.0026	.001125	.000225
		(1) → (2) → (3)			
1					
26	76	86	96	101	111
.0049	.0174	.0084	.0075	.006025	.005125
			(4) → (5) → (6) → (7)		
2					
39	89	99	109	114	124
.000343	.012843	.003843	.002943	.001468	.000568
3					
3					
52	102	112	122	127	137
.000024	.01250	.003524	.002624	.001149	.000249
4					
65	115	125	135	140	150
.00000168	.01250	.00350	.00260	.001126	.00022668

Fig. 2. Combining Subsystems 1 and 2 with 3

$$UR_{ij} = (1-R_i)+(1-R_j)-(1-R_i)(1-R_j) \tag{22}$$

$$C_{ij} = C_i + C_j \tag{23}$$

After all subsystems have been combined, the allocation is found by examining the last combination to find the entry that meets the required reliability and has the lowest cost, then working back through each subsystem to find the allocation for that subsystem. For this example the final allocation is given in Table II.

1, 2 & 3					
Sequence Number	1	2	3	4	5
Cost	76	86	96	99	109
Unreliability	.0174	.0084	.0075	.003943	.002943
		(1) → (2)	→ (3)	→ (4)	
1					
34	110	120	130	133	143
.0064	.0238	.0148	.0139	.01024	.00934
4		(5)			
2					
51	127	137	147	150	160
.000512	.017912	.008912	.008012	.004356	.003455
3					
68	144	154	164	167	177
.00004096	.01744	.00844	.00764	.003883	.002983

Fig. 3 Combining Subsystems 1, 2, and 3 with 4

Table II
Final Allocation of Example

System Cost 137
System Reliability .9911

Subsystem	Cost	Unreliability	Reliability
1	30	.001	.999
2	30	.0025	.9975
3	26	.0049	.9951
4	51	.00051	.99949

There are two refinements for this procedure that will reduce the time required. These improvements restrict the number of entries that must be examined in order to find the next element in the dominating sequence.

(1) A row or column for which the unreliability is greater than the unreliability of the present entry in the dominating sequence need not be examined to find the next element in the dominating sequence.

(2) If the present entry in the dominating sequence is in column i and row j, then the next entry will either be in column $k \leq i$ or row $l \leq j$.

A computer listing of a program to make these calculations is given in Appendix III.

V. CONCLUSIONS

After each of the allocation models had been developed, the vehicle with which to compare them had to be selected. The decision was made to compare them using a system composed of 20 subsystems. The component reliabilities and costs for each of these 20 subsystems were selected at random. The component reliabilities were selected from the uniform distribution from 0.5 to 1.0. The component costs were selected from the uniform distribution from 0.0 to 1000.0. (See Appendix IV) The system reliability requirement was selected to be .998.

RESULTS

As seen from Table IV, the solution given by the dynamic programming method was optimal. The system cost from the allocation made by the Lagrange multiplier method was within 0.5 per cent of the optimal solution. The solution given by the mathematical programming algorithm was within 0.6 per cent of the optimal solution. From this it is seen that all of the allocation algorithms give solutions that are either optimal or close to optimal.

According to the theory of dynamic programming, an optimal solution is guaranteed, so the result from this part

of the problem was as expected.

Now consider the result of the Lagrange multiplier algorithm. The theory guarantees that the allocation given by this method is optimal. However, it is an optimal solution of a reliability allocation problem where the desired system reliability is equal to .99808056. A gap exists around the area that gives a system reliability equal to .998. The Lagrange multiplier technique is unable to find the optimal solution in this area. This gap is due to a nonconcavity in the function of optimum payoff in terms of resource constraints (Ref 4:407).

The mathematical programming method is based on a heuristic argument and does not guarantee an optimal solution. The solution given, which is within 0.6 per cent of the optimal solution is as good as expected.

The major disadvantage of the dynamic programming algorithm is the large computer memory required to make the calculations. In the problem considered here, at stage 19 the optimal sequence contained 667 entries. Each entry requires a storage space for cost and one for unreliability. In addition, if the program is to print out a final solution, enough space is required to store the location of each entry in the optimal sequence for each stage in dynamic programming process. In this problem a matrix of more than 20 x 667

is required to store the location of each entry in the optimal sequence for each stage. The total memory required for the dynamic programming algorithm was 22695 bits. This is approximately 50 times the memory required for either the Lagrange multiplier technique or the mathematical programming technique.

Table III
Component Reliability and Cost
for Each Subsystem in the Representative System

Subsystem	Cost	Reliability
1	593	.50000069
2	641	.50001587
3	750	.50036501
4	253	.50839523
5	824	.69309029
6	956	.94107663
7	984	.64476239
8	641	.82953494
9	752	.47390355
10	285	.82398152
11	554	.95175489
12	744	.88622^37
13	101	.88311443
14	314	.81163169
15	228	.66752880
16	245	.8531637
17	635	.62273432
18	599	.82288922
19	767	.92645199
20	630	.80839568

Table IV
Final Allocation for Each Method

Subsystem	Dynamic Programming		Lagrange Multipliers		Mathematical Programming	
	Cost	Reliability	Cost	Reliability	Cost	Reliability
1	7709	.99987793	7709	.99987793	7709	.99987793
2	7692	.99975595	7692	.99975595	7692	.99975595
3	9000	.99975798	9000	.99975798	9000	.99975798
4	3542	.99995184	3289*	.99990205*	3289*	.99990205*
5	6592	.99992128	6592	.99992128	6592	.99992128
6	3824	.99998794	3824	.99998794	3824	.99998794
7	7872	.99974640	7872	.99974640	7872	.99974640
8	3205	.99985605	3205	.99985605	3205	.99985605
9	7520	.99983634	7520	.99982634	7520	.99982634
10	1710	.99997026	1710	.99997026	1710	.99997026
11	1662	.99988644	1662	.99988644	1662	.99988644
12	2976	.99983241	3720*	.99998093*	3720*	.99998093*
13	606	.99999744	505*	.99997818*	606	.99999744
14	1884	.99995533	1884	.99995533	1884	.99995533
15	2052	.99995036	2052	.99995036	2052	.99995036
16	1470	.99998997	1470	.99998997	1470	.99998997
17	5715	.99984518	5715	.99984518	5715	.99984518
18	3594	.99996913	3594	.99996913	3594	.99996913
19	3068	.99997073	3068	.99997073	3068	.99997073
20	3780	.99995051	3780	.99995051	3780	.99995051
Total	85473	.99800140	85863	.99808056	85964	.99809980

* Indicates subsystem in which allocation by given method differs from optimum.

As a further comparison of the relative length of computation for the three algorithms, consider the computer time required for each. The dynamic programming algorithm required 9 minutes and 55 seconds, the mathematical programming algorithm required 50 seconds, and the Lagrange multiplier technique required only 36 seconds.

One other area of comparison is the area of flexibility. All three of the allocation methods could, with little effort, be made to handle the situation where the cost function is nonlinear.

The change required in the Lagrange multiplier technique would be to substitute the new cost function in equation (17). The change required of the dynamic programming algorithm would be to read into the computer the new reliability cost sequence for each stage. The change in the mathematical programming algorithm would require a method for calculating the increase in system cost by adding one component at the ith subsystem similar to the method for calculating the increase in reliability as given in equation (20).

SUMMARY

From the results of the problem considered here, it is felt that the Lagrange multiplier technique is best. Although the solution obtained by the use of dynamic programming was optimal, it is felt that the large time requirement offsets this fact.

VI. RECOMMENDATIONS FOR FURTHER STUDY

The next direction to be followed in this course of study should be to consider the problem of maximizing the reliability of the system subject to multiple constraints. Bellman and Drefus (Ref 2) and Proschan and Bray (Ref 14) have examined the dynamic programming aspects of the problem involving multiple constraints. Everett (Ref 4) has formed the basis for study in this area using Lagrange multipliers. Koelesar (Ref 11) has considered the solution of this problem by the use of linear programming.

It is felt that each of these areas is worth further examination.

BIBLIOGRAPHY

1. Aris, Rutherford. Discrete Dynamic Programming. New York: Blaisdel Publishing Company, 1964.

2. Bellman, Richard E. and Stuart E. Dreyfus. Applied Dynamic Programming. Princeton, New Jersey: Princeton University Press, 1962.

3. Dantzig, George B. Linear Programming and Extensions. Princeton, New Jersey: Princeton University Press, 1963.

4. Everett, Hugh III. "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources." Operations Research, 10:399-417 (May-June, 1963).

5. Glover, Fred. "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem." Operations Research, 13:879-919 (November-December, 1965).

6. Heyne, J. B. and G. H. Sandler. "System and Subsystem Peliability." in Reliability Engineering for Electronic Systems. Edited by Richard H. Meyers, et al. New York: John Wiley and Sons, Inc., 1964

7. Hildebrand, F. B. Methods of Applied Mathematics. New York: Prentice Hall, Inc., 1952.

8. Hillier, Frederick S. and Gerald J. Lieberman. Introduction to Operations Research. San Francisco: Holden-day, Inc., 1967.

9. Hinely, J. T. and B. F. Shelley. Method of Allocating Reliability Goals to Aircraft Subsystems and Components. Aerospace Reliability and Maintainability Conference. AIAA/SAE/ASME, 1963.

10. Kettelle, John D. Jr. "Least Cost Allocation of Reliability Investment." Operations Research, 10:249-265 (March-April, 1962).

Bibliography (Contd)

11. Kolesar, Peter J. "Linear Programming and the Reliability of Multicomponent Systems." Naval Research Logistics Quarterly, 14:317-327 (September, 1967).

12. Mroz, G. J. and E. A. Yerman. Reliability Apportionment and Optimization Techniques. Technical Memorandum 31-024-43-ORA-10 Downey, California: Autonetics, 1963.

13. Nemhauser, George L. Introduction to Dynamic Programming. New York: John Wiley and Sons, Inc., 1966.

14. Proschan, Frank and T. A. Bray. "Optimum Redundancy Under Multiple Constraints." Operations Research, 13:800-814 (September-October 1965).

15. Zelan, Marvin, editor. Statistical Theory of Reliability. Madison, Wisconsin: University of Wisconsin Press, 1963.

APPENDIX I

Lagrange Multiplier Method

This appendix contains a description of the program that was used for the Lagrange multiplier allocation method. The program was written using Fortran IV for the IBM 7044/7094 system.

Table V
Symbols Used in the Lagrange Multiplier Method

Symbol	Description of Meaning
Input Variables	
RS	The required system reliability.
ML	Number of subsystems in the system. ML must be less than or equal to 20.
RC(I)	Reliability of the components used in the Ith subsystem.
CC(I)	Cost of the components used in the Ith subsystem.
Program Variables	
ZL	The Lagrange multiplier.
US	Required system unreliability.
UC(I)	Unreliability of components used in the Ith subsystem.
N(I) K(I)	Number of components allocated to the Ith subsystem.
N K	Matrix that contains the number of components of each subsystem.
XZL	Increment to find the Lagrange multiplier that gives a system reliability less than RS.
L	Level of the Lagrange multiplier increment.

Table V (Contd)

Symbol	Description of Meaning
Program Variables	
DZ(L)	Lagrange multiplier increment. DZ(1) = XZL/10 DZ(2) = XZL/100.
TZL	Lower limit of the Lagrange multiplier.
Program Subroutines	
BEST(ZL,K)	Program to calculate the optimal number of components for each subsystem, K, given the Lagrange multiplier, Z.
URS(K)	Program to calculate the system unreliability, given K.
COST(K)	Program to calculate the system cost given K.
COSTS(I,J)	Cost of the Ith subsystem when it consists of J components.
R(I,J)	Reliability of the Ith subsystem when it consists of J components.

PROGRAM SUMMARY

After the input data is received, the program starts an iterative procedure (statements 7 through 12) to find a Lagrange multiplier that will give a system reliability less than required. This is done because the system reliability is a monotonic decreasing function of the Lagrange multiplier (Ref 4:406).

The next section (statements 15 through 17) define the limits to be searched and the Lagrange multiplier increment.

The third section compares the allocation for two

different values of the Lagrange multiplier. Statements 25 through 29 check for one change in the allocation. If no changes occur, ZL is decreased and the search is made again. If a change does occur, the five statements after statement 30 search for a second change. If two changes have occurred, statements 40 through 50 reduce the interval and the machine is sent back to find the level where only one change has occurred. If two changes have not occurred, the machine prints out the result, then continues the search.

```
$JOB            0,2,2000        69-356  LT SPIVEY AFIT-SE
$EXECUTE        IBJOB
$IBJOB
$IBFTC MAIN
      DIMENSION N(20),RC(20),CC(20),UC(20),DZ(5),X(5),K(20)
      COMMON CC,UC,ML
    1 READ (5,201) RS,ML
    2 READ (5,202 )   (RC(I),I=1,ML)
    3  READ(5,203) (CC(I),I=1,ML  )
      US = 1.0- RS
    5 DO 6 I = 1,ML
    6 UC(I) = 1.0 -RC(I)
    7 ZL = 0.0
      NI = ALOG(US/FLOAT(ML))/ALOG(UC(1))
      XZL = (UC(1)**NI)*(-ALOG(UC(1)))/(CC(1)*(1.0-UC(1)**NI))
    8 ZL = ZL +XZL
    9  CALL BEST (ZL,K)
   11 IF(UPS(K) .GE. US) GO TO 15
   12 GO TO 8
C     PREVIOUS STATEMENTS CALCULATE THE INITIAL LAGRANGE
C     MULTIPLIER,ZL, AND THE INCREMENT TO BE ADDED TO FOR
C     SUBSEQUENT MULTIPLIERS.
   15 TZL = ZL - XZL
      L=1
      X(L) = 1.0
   17 DZ(L) = .1*XZL
      GO TO 100
   20 ZL = ZL - DZ(L)
   25 CALL BEST (ZL,K)
      I = 6
   26 IF (N(I) .NE.K(I))GO TO 30
   27 IF(I.GE.ML) GO TO 50
   28 I = I+1
   29 GO TO 26
C     STATEMENTS 26 THROUGH 29 CHECK FOR A CHANGE IN
C     THE ALLOCATION.
   30 I = I+1
      IF(I .GT. ML) GO TO 100
      IF(N(I) .NE. K(I)) GO TO 40
      GO TO 30
C     STATEMENTS AFTER 30 CHECK FOR A 2ND CHANGE IN THE
C     ALLOCATION.
   40 L = L+1
      X(L) =ZL
      ZL = ZL + DZ(L-1)
      DZ(L) = .1*DZ(L-1)
      GO TO 20
   50 IF( ZL .LE. X(L)) GO TO 52
C     IF 2 CHANGES OCCUR, STATEMENTS 40 THROUGH 50 REDUCE THE
C     INCREMENT AND SEND THE COMPUTER BACK TO CHECK THE
C     INTERVAL WHERE THE 2 CHANGES OCCURRED.
      GO TO 20
   52 IF (L .GT. 1) L = L-1
   53 IF(ZL .LT. TZL ) CALL EXIT
      GO TO 20
```

```
      100 WRITE (6,204)
      101 WRITE (6,205) (K(I),I = 1 ,ML)
      102 WRITE (6,206)
          C = COST (K)
          REL = 1.0 -UNRS(K)
      103 WRITE (6,207) ZL,C,REL
          WRITE (6,208)
          DO 104 I = 1,ML
          CS = COSTS (I,K(I))
          RT = R(I,K(I))
      104 WRITE (6,301)I,CS,RT
          DO 105 I = 1,ML
      105 M(I) = K(I)
          GO TO 20
      201 FORMAT (F12.6,I3)
      202 FORMAT (5F10.6)
      203 FORMAT (5F10.0 )
      204 FORMAT(1HI6X,41HTHE NUMBER OF COMPONENTS IN EACH STAGE IS)
      205 FORMAT (1H0,6X,5(I3,4X))
      206 FORMAT (7X,6HALGOA,10X,4HCOST, 10X,11HRELIABILITY )
      207 FORMAT (5X, F10.6,4X,F10.0,5X,F10.6 )
      300 FORMAT(1H0,10X,5HSTAGE,6X,4HCOST,4X,11HRELIABILITY)
      301 FORMAT (1H0,11X,I3,2X,F9.0,4X,F10.6)
          END
```

```
      SUBROUTINE TEST(Z,K)
C     SUBPROGRAM TO FIND THE OPTIMUM NUMBER OF REDUNDANT
C     COMPONENTS.
      DIMENSION K(20) ,UC(20), CC(20)
      COMMON CC, UC, ML
      U(N,U,C)=ALOG(1.0-U**N)-Z*C*FLOAT(N)
    1 DO 10 I= 1,ML
    2 NN=(ALOG(Z*CC(I))-ALOG(Z*CC(I)-ALOG(UC(I))))/ALOG(UC(I))
    3 ML = NN
    4 NU = ML +1
      IF (ML .LT. 1 ) GO TO 8
    5 IF(U(NU, C(I),CC(I)) .GT.U(ML,UC(I),CC(I))) GO TO 8
      K(I) = ML
      GO TO 10
    8 K(I) = NU
   10 CONTINUE
      RETURN
      END
```

```
C$IFTC SUB2
C SUBPROGRAM TO FIND TOTAL SYSTEM COST
      FUNCTION COST(N)
      DIMENSION N(20) ,CC(20) ,UC(20)
      COMMON CC,UC,NL
      COST = 0.0
      DO 5 I = 1,NL
    5 COST = COST +CC(I)*FLOAT(N(I))
      RETURN
      END


C$IFTC SUB3
C SUBPROGRAM TO FIND SYSTEM UNRELIABILITY
      FUNCTION URS(N)
      DIMENSION N(20) ,UC(20) , CC(20)
      COMMON CC,UC,NL
    1 RS = 1.0
    2 DO 5 I = 1,NL
    5 RS = RS*(1.0-UC(I)**N(I))
      URS = 1.0-RS
      RETURN
      END


C$IFTC SUB4
C PROGRAM TO CALCULATE SUBSYSTEM COST
      FUNCTION COSTS(I,J)
      DIMENSION CC(20),RC(20),N(20)
      COMMON CC,RC,NL
      COSTS = CC(I)*FLOAT(J)
      RETURN
      END


C$IFTC SUB5
C     PROGRAM TO CALCULATE SUBSYSTEM RELIABILITY
      FUNCTION R(I,J)
      DIMENSION CC(20), UC(20)
      COMMON CC,UC,NL
      R = 1.0 -UC(I)**J
      RETURN
      END
      END
```

APPENDIX II

Mathematical Programming Method

This appendix contains a description of the program that was used for mathematical programming. The program was written in Fortran IV for the IBM 7044/7094 operating system.

Table VI
Symbols Used in Mathematical Programming

Symbol	Description of Meaning
Input Variables	
RS	The required system reliability.
ML	The number of subsystems in the system.
RC(I)	Reliability of components in the Ith subsystem.
CC(I)	Cost of Components in the I subsystem.
Program Variables	
N(I)	Number of components in the Ith subsystem.
N1(I)	N(I) + 1, N1(J) = N(J) for all J unequal to I.
A(I)	Increase in reliability per unit cost by adding one component to the Ith subsystem.
Program Subroutines	
RST(N)	System reliability given N(I) components at each stage.
R(I,J)	Reliability of the Ith subsystem when it consists of J components.
COST(N)	System cost given N(I) components at each stage.
C(I,J)	Cost of the Ith subsystem when it contains J components.

PROGRAM SUMMARY

After the input data has been read, the five statements before statement 10 calculate the initial solution. The next section has the machine print out the present solution. The third section (statements 13 through 20) calculate A(I), the increase in reliability per unit cost for the Ith subsystem. Statements 25 through 31 add one component to the subsystem for which the value of A(I) is the largest. Next, the machine is sent back to statement 12 to check if the present solution gives a system reliability as large as required. If it does, the results is printed out and the program ends. If it does not, the procedure is repeated until the solution does meet the reliability requirement.

```
[illegible]            [illegible]            [illegible]  LT [illegible]
[illegible]            [illegible]
[illegible]
[illegible]
C [illegible] T. [illegible] LINEAR PR[illegible]
      DIMENSION RC(20),CC(20),N(20),NI(20),A(20)
      [illegible] CC,RC,ML
    1 READ (5,201) RS,ML
    2 READ (5,202) (RC(I),I = 1,ML)
    3 READ (5,203) (CC(I),I =1,ML)
C     INPUT.  RS = DESIRED SYSTEM RELIABILITY.  ML = [illegible]
C     OF SUBSYSTEMS.
C     RC(I) = RELIABILITY OF COMPONENTS USED IN SUBSYSTEM I.
C     CC(I) = COST OF COMPONENTS USED IN SUBSYSTEM I.
      DO 10 I = 1,ML
C     LOOP TO CALCULATE INITIAL SOLUTION.
      N(I) = ALOG(1.0-RS)/ALOG(1.0-RC(I))
      IF (N(I) .LT.0) N(I) = 1
      NI(I) = N(I)
   10 CONTINUE
   12 IF(RST(N) .GE.RS) GO TO 100
      RR = RST(N)
      CK = COST (N)
      WRITE (6,204) RR,CK
      WRITE (6,205) (N(I),I = 1,ML)
C     INTERIM OUT PUT TO SHOW PROGRESS.
   13 DO 20 I = 1,ML
C     LOOP TO CALCULATE A(I).
      NI(I) = N(I) + 1
      A(I) = (RST(NI) - RST(N))/CC(I)
   20 NI(I) = N(I)
      TEMP = A(1)
      L = 1
   25 DO 30 I = 2,ML
C     LOOP TO FIND LARGEST A(I).
      IF (TEMP .GE. A(I)) GO TO 30
      TEMP = A(I)
      L = I
   30 CONTINUE
   31 N(L) = N(L) + 1
C     STATEMENT TO ADD ONE MORE COMPONENT TO SUBSYSTEM WITH
C     THE LARGEST A(I).
      GO TO 12
  100 WRITE (6,210)
C     OUTPUT
      RR = RST (N)
      CK = COST(N)
C     N IS THE MATRIX CONTAINING THE NUMBER OF COMPONENTS IN
C     EACH SUBSYSTEM.
      WRITE (6,211) CK,RR
      WRITE (6,212)
  210 FORMAT(1H1,10X,4HCOST,5X,11HRELIABILITY)
  211 FORMAT(4X,F10.0,5X,F10.6)
      DO 110 I = 1,ML,2
      II=I+1
```

```
      [illegible]
      R1 = R(I,N(I))
      R2 = R(II,N(II))
      C1 = C(I,N(I))
      C2 = C(II,N(II))
      WRITE (6,219) I,R1,C1,II,R2,C2
  111 CONTINUE
  201 FORMAT(F12.6,I3)
  202 FORMAT(5F10.8)
  203 FORMAT(5F10.0   )
  204 FORMAT (1H0,4X,F10.8,4X,F10.1      )
  205 FORMAT (1H0,4(2X,I3))
  218 FORMAT(1H03X,2(9HSUBSYSTEM2X,11HRELIABILITY7X,4HCOST10X))
  219 FORMAT(3X,2(4X,I3,4X,F10.8,4X,F10.0))
  220 FORMAT (5F16.8)
      CALL EXIT
      STOP
      END
```



```
$IBFTC SUB1
C     PROGRAM TO CALCULATE SYSTEM RELIABILITY.
      FUNCTION PST(N)
      DIMENSION CC(20),RC(20), N(20)
      COMMON CC, RC, ML
      PST = 1.0
      DO 10 I = 1,ML
      PST = PST *R(I,N(I))
   10 CONTINUE
      RETURN
      END
```

```
$IBFTC SUB2
C     PROGRAM TO CALCULATE SUBSYSTEM RELIABILITY.
      FUNCTION R(I,J)
      DIMENSION CC(20),RC(20)
      COMMON CC,RC,ML
      R = 1.0 - (1.0 - RC(I))**J
      RETURN
      END
```

[illegible]

```
$IBFTC [illegible]
C       PROGRAM TO CALCULATE SYSTEM COST
        FUNCTION COST( )
        DIMENSION  CC(20), [illegible](20),[illegible](20)
        COMMON [illegible], [illegible] L
        COST = [illegible]
        DO 10 I = 1,ML
     10 COST = COST +CC(I)*FLOAT(M(I))
        RETURN
        END
```

```
$IBFTC SUB4
C       PROGRAM  TO CALCULATE SUBSYSTEM COST.
        FUNCTION C(I,J)
        DIMENSION CC(20),SC(20),M(20)
        COMMON CC,BC,ML
        C = CC(I) * FLOAT(J)
        RETURN
        END
$DATA
        END
```

APPENDIX III

Dynamic Programming Method

This appendix contains the program description for the dynamic programming algorithm. The program was written in Fortran IV for use on the IBM 7044/7094 operating system.

Table VII
Symbols used in the Dynamic Programming Method

Symbol	Description of Meaning
Input Variables	
RS	The required system reliability.
ML	Number of subsystems in the system.
RC(I)	Reliability of the components used in the Ith subsystem.
CC(I)	Cost of the components used in the Ith subsystem.
Program Variables	
U(I)	Unreliability of components used in the Ith subsystem.
U(I,J) C(I,J)	Unreliability-cost sequence for the Ith subsystem.
M	Number of the subsystem that is being that is being combined with the combination of the M-1 previous subsystems.
YU(I) YC(I)	Unreliability-cost sequence for the Mth subsystem.
XU(I) XC(I)	Unreliability-cost sequence for the combination of the M-1 previous subsystems.
AU(I) AC(I)	Dominating sequence for the combination of the Mth subsystem with the M-1 previous subsystems.
L(M,I)	Location of the two elements that were combined to give the Ith entry in the dominating sequence for the M subsystem combination.

Table VII (Contd)

Symbol	Description of Meaning
Program Variables	
EUX, ECX	Candidate for the next entry in the dominating sequence.
EUY, ECY	Candidate for the next entry in the dominating sequence.
Program Subroutines	
OUTPUT(N)	Subprogram to print out the dominating sequence, which is of length N, resulting from the combination of the Mth subsystem with the M-1 previous subsystems.
ANSWER(N)	Subprogram to print out the final allocation given the dominating sequence of length N after all subsystems have been combined.

PROGRAM SUMMARY

After the data has been read, statements 11 through 20 calculate the base requirements for each subsystem. Statements 21 through 39 calculate the reliability-cost sequence for each subsystem and write out this information. Statements after 39, through 40 make a block transfer of the reliability-cost sequence for the first subsystem into the location (XU, XC) for the dominating sequence of the M-1 previous subsystems. Since M equals 2, this combination is the reliability-cost sequence of subsystem 1.

The next section (statements 50 through 55) is a block transfer to move the reliability-cost sequence of the Mth subsystem into the location to be combined with the combination of the M-1 previous subsystems.

The following section (statements 60 through 110) find the dominating sequence, one element at a time, of the combination of the Mth subsystem and the M-1 previous subsystems and write out this sequence.

The last section makes a block transfer of the dominating sequence obtained in the previous section into the location to be combined with the next subsystem. It also increments M to the next subsystem and tests to see if all subsystems have been combined. If they have, the final allocation is printed out. If they have not, the machine is sent back to statement 50 to continue the procedure.

```
 [illegible]

*JOB            0,8.4000         65.266  LT SPIVEY [illegible]
$EXECUTE        IBJOB
$IBJOB
$IBFTC MAIN
      COMMON Y,US,NC,L
      COMMON/B1/ U,C,M,ML
      DIMENSION RC(20),CC(20),UC(20),NL(20),U(20,10),C(20,10)
      DIMENSION XU(900),XC(900),ZU(900),ZC(900),YU(10),YC(10)
      DIMENSION L(20,900) .
    1 READ (5,201) RS,ML
    2 READ (5,202 )  (RC(I),I=1,ML)
    3 READ(5,203) (CC(I),I=1,ML  )
C     CC(I) = COST OF COMPONENTS USED IN SUBSYSTEM I.
C     RC(I) = RELIABILITY OF COMPONENTS USED IN SUBSYSTEM I.
      US = 1.0- RS
    5 DO 10 I= 1,ML
   10 UC(I) = 1.0-RC(I)
   11 DO 20 I = 1,ML
      K=1
   13 X=UC(I)**K
   14 IF(US-X) 15,16,16
   15 K=K+1
C     LOOP TO CALCULATE THE NUMBER OF COMPONENTS IN EACH
C     SUBSYSTEM REQUIRED TO MEET THE BASE REQUIREMENT.
      GO TO 13
   16 NL(I) = K
   20 U(I,1) =X
   21 DO 35 I=1,ML
C     LOOP TO CALCULATE THE RELIABILITY COST SEQUENCE FOR
C     EACH SUBSYSTEM.
      Y = NL(I)
      C(I,1)=Y*CC(I)
      DO 35 J = 2,10
      U(I,J)= U(I,J-1)*UC(I)
   35 C(I,J) =C(I,J-1)+CC(I)
      NI = 10
      DO 39 I = 1,ML
      WRITE (6,210) I
C     LOOP TO PRINT OUT THE RELIABILITY COST SEQUENCE FOR
C     EACH SUBSYSTEM.
      DO 39 J = 1,10,2
      JJ = J+ 1
      RIJ = 1.0 -U(I,J)
      RIJJ = 1.0 - U(I,JJ)
      WRITE (6,211) J,RIJ,C(I,J),JJ,RIJJ,C(I,JJ)
   39 CONTINUE
      DO 40 I=1,10
      XU(I) = U(1,I)
   40 XC(I) = C(1,I)
      M=2
      NI = 10
   50  DO 55 I = 1,10
      YU(I) = U(M,I)
   55 YC(I) = C(M,I)
      US = US/FLOAT (ML + 1 - M)
```

[illegible]

```
C     [illegible] CALCULATE THE [illegible]ILITY OF EACH
C     [illegible] THAT [illegible]ILL SUPPORT THE SYSTEM REQUIREMENT.
C     [illegible](I) = XU(I) + YU(I) - XU(I)*YU(I)
      [illegible](I) = XC(I) + YC(I)
C     [illegible] TO FIND INITIAL ENTRY IN THE DOMINATING
C     [illegible] STACK.
      L([illegible]) = [illegible]
      [illegible] = [illegible]
      J = 1
      [illegible] = 1
   [illegible] [illegible] = XU(I[illegible])
      [illegible] = [illegible]([illegible])
      [illegible] = [illegible]([illegible])
      [illegible]X = [illegible]
      [illegible]Y = [illegible]
C     [illegible] 7[illegible] THROUGH 80 EXAMINE ALL ROWS LESS THAN OR
C     [illegible] TO J TO FIND A CANDIDATE FOR THE NEXT ELEMENT
C     IN THE [illegible] SEQUENCE.
   [illegible] [illegible] = 1,[illegible]
      I[illegible] (XU(K) .[illegible]. [illegible]) GO TO [illegible]1
      D[illegible] [illegible] L1 = 1,J
      IF (YU(L1).GE. [illegible]) GO TO 80
      [illegible] = X[illegible](K) + Y[illegible](L1) - X[illegible](K)*YU(L1)
      IF([illegible] .[illegible].[illegible]) GO TO 80
      [illegible] = XC(K) + YC(L1)
      IF([illegible] .LT.[illegible]) GO TO 80
      IF(([illegible].[illegible]).[illegible].(LC1.EQ.LCX)) GO TO 80
      [illegible]X = [illegible]
      [illegible]X = [illegible]
      LLX = [illegible] + L1
   [illegible] CONTI[illegible]
   8[illegible] CONTI[illegible]
C     [illegible] 8[illegible] THROUGH 90 EXAMINE ALL COLUMNS LESS THAN OR
C     [illegible] I TO FIND A CANDIDATE FOR THE NEXT ELEMENT IN
C     THE [illegible] SEQUENCE.
      [illegible] = 1,I
      IF (XU([illegible]) .[illegible]. [illegible]) GO TO 91
      [illegible] L1 = 1,[illegible]
      IF (YU(L1) .[illegible]. [illegible]) GO TO 90
      [illegible] = X[illegible](K) + YU(L1) - XU(K)*YU(L1)
      IF([illegible] .[illegible].[illegible]) GO TO 90
      [illegible] = XC(K) + [illegible](L1)
      IF([illegible] .LT.[illegible]) GO TO 90
      IF(([illegible].[illegible]).[illegible].(LC1.[illegible].LCY))GO T[illegible] [illegible]
      [illegible]Y = [illegible]
      [illegible] = [illegible]
      LLY = [illegible] + L1
   [illegible] CONTI[illegible]
   [illegible]1 CONTI[illegible]
C     [illegible] THROUGH 10[illegible] SELECT THE NEXT ENTRY IN
C     THE [illegible] SEQUENCE, [illegible], FROM EITHER LCX OR [illegible].
      I[illegible] = I[illegible]+ 1
      I[illegible]( [illegible] .LT. [illegible]X ) GO TO 10[illegible]
      I[illegible](([illegible] .LT.[illegible]) .[illegible].(LCY.[illegible].LCX)) GO TO 100
      [illegible]([illegible]) = [illegible]X
```

```
[illegible]

      [illegible](I[illegible]) = [illegible]X
      L([illegible],I[illegible]) = LLX
      GO TO 110
  1[illegible] [illegible](I[illegible]) = [illegible]Y
      [illegible](I[illegible]) = [illegible]Y
      L([illegible],I[illegible]) = LLY
  11[illegible] IF([illegible](I[illegible]) .[illegible].[illegible]) GO TO 120
C     [illegible] TO TEST IF THE [illegible] IN ENTRY IN THE
C     [illegible] [illegible]ATING [illegible] WILL SUPPORT THE RELIABILITY
C     [illegible] FOR STAGE [illegible].
      I = L([illegible],I[illegible])/100
      J = L([illegible],I[illegible]) - 100*I
      GO TO 6[illegible]
  120 CALL OUTPUT(IN)
      [illegible] = [illegible]+ 1
      NI = I[illegible]
      DO 12[illegible] I = 1,NI
      XU(I) = AU(I)
  12[illegible] XC(I) = AC(I)
C     [illegible] TRANSFER TO SET XU(I)= AU(I) PRIOR TO [illegible]
C     X[illegible](I) WITH [illegible](I) TO FORM THE NEXT [illegible] [illegible].
      IF([illegible] .LT.[illegible]L) GO TO 9[illegible]
C     TEST TO [illegible] IF ALL STAGES HAVE BEEN [illegible].
      CALL ANSWER ([illegible]I)
  2[illegible] [illegible]AT (F12.6, I3)
  2[illegible] FORMAT (5F10.6)
  2[illegible] FORMAT (5F10.[illegible])
  2[illegible] [illegible](1[illegible],2X,[illegible],I3,2X,2([illegible]LOCATION,2X,11H[illegible]
     1,4X,[illegible] ,[illegible]X,))
  21[illegible] FORMAT( 14X,2(I3,[illegible]X,F1[illegible].[illegible],2X,F1[illegible].[illegible],6X))
  3[illegible] FORMAT(1[illegible]X,1[illegible] STATEMENT ,I[illegible] )
  [illegible] FORMAT([illegible]X,2(I[illegible],4X,F1[illegible].[illegible],2X,F1[illegible].[illegible],6X))
      RETURN
      END


[illegible]
      SUBROUTINE OUTPUT ([illegible]I)
      DIMENSION AU([illegible]),AC(500),L(2[illegible],[illegible])
      COMMON [illegible],AU,A[illegible],L
      WRITE (6,206)[illegible]
      WRITE (6,2[illegible]7)
      DO 1[illegible] I = 1,[illegible]I,2
      II = I+1
      Z = 1. - A[illegible](I)
      Z1 = 1.[illegible] - AC(II)
      WRITE (6,2[illegible]6) I,Z,[illegible](I),L([illegible],I),II,Z1,[illegible](II),L([illegible],II)
  1[illegible] CONTINUE
  2[illegible] FORMAT ([illegible],[illegible]X,[illegible] NUMBER OF [illegible] [illegible] IS [illegible])
  206 F[illegible]([illegible]X,2(I[illegible],5X,F1[illegible].[illegible],1X,F1[illegible].[illegible],6X,I[illegible],[illegible]))
  2[illegible]7 [illegible]AT(6X,2([illegible]LOCATION,2X,11H[illegible]LIABILITY,5X,[illegible],2X,
     1[illegible]NEXT LOCATION,5X))
      RETURN
      END
```

```
[illegible]

$IBFTC [illegible]
      SUBROUTINE [illegible] (N,L)
      DIMENSION AC(5[illegible]),L(20,5[illegible]),RA(20),CA(20),AC(5[illegible])
      DIMENSION R(20,10),C(20,10)
      COMMON [illegible],AJ,AC,L
      [illegible] /[illegible]/ R,C,[illegible],L
  [illegible] DO 225 I = 1,N
      IF(A[illegible](I) .LE. [illegible]) GO TO 226
  225 CONTINUE
  226 N = [illegible]
      K = 1
  227 J = L(N,I) /100
      N = L(N,I) -J*100
      RA(K) = 1.0 -R(N,K)
      CA(K) = C(N,K)
      I = J
      K = K-1
      IF(K .[illegible] .2) GO TO 227
      RA(1) = 1.0-R(1,J)
      CA(1) = C(1,J)
      WRITE (6,300)
      Z = 1.0 - AC(N)
      WRITE (6,301) Z,AC(N)
      WRITE (6,302)
      DO 240 I = 1, L,2
      II = I + 1
  240 WRITE (6,303) I,RA(I),CA(I),II,RA(II),CA(II)
  300 FORMAT(1H1,[illegible]X,18HSYSTEM RELIABILITY,5X,11HSYSTEM COST )
  301 FORMAT(1H [illegible]X,F10.8,1[illegible]X,F10.[illegible] )
  302 FORMAT(1H0,4X,2([illegible]SYSTEM,5X,11HRELIABILITY,6X,4HCOST [illegible]X))
  303 FORMAT(2(7X,I3,7X,F10.8,3X,F10.0,2X))
      RETURN
      END
$DATA
      END
```

NOT REPRODUCIBLE

APPENDIX IV

Random Number Generator

This appendix describes the program that was used to generate the component cost and reliability for each subsystem. The program was written in the Forgo version of Fortran II. In the particular compiler used, the arctangent subroutine had been replaced with a random number generator. This program was written for the IBM 1620.

Table VIII
Symbols Used
in Random Number Generator

Symbol	Meaning
R(I)	Component reliability.
C(I)	Component cost.
X	Temporary location.

[illegible] 7-176?-,

```
C   C   [illegible] ZACK SPIVEY THESIS    NUMBER SOURCE
C       THIS PROGRAM GENERATES 20 RANDOM NUMBERS BETWEEN 0.5 AND
C       1.0 AND 20 RANDOM NUMBERS BETWEEN 0.0 AND 1000.0, THEN
C       PUNCHES THEM OUT 5 TO A CARD.
        DIMENSION R(20), C(20)
        DO 10 I = 1,20
        X = RAN(0.0)
        IF(X- .5) 8,9,9
      8 X = X + .5
      9 R(I) = X
     10 CONTINUE
        DO 20 I = 1,20
     20 C(I) = 1000. * RAN(0.0)
        PUNCH 201,(R(J), J = 1,20)
        PUNCH 202,(C(J), J = 1,20)
    201 FORMAT(5F10.8)
    202  FORMAT(5F10.0)
        END
```

NOT REPRODUCIBLE

GRE/EE/69-2

VITA

William Zachary Spivey was born on 30 March 1945 in Ruston, Louisiana. He was graduated from Delhi High School in Delhi, Louisiana in 1963. He attended Louisiana Polytechnic Institute and upon graduation in 1968, he was awarded a Bachelor of Science in Mechanical Engineering and a commission in the United States Air Force.

Permanent Address: Rt 2, Box 56
Ruston, Louisiana 71270

This thesis was typed by Bobbie Thompson

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

Air Force Institute of Technology (AFIT-SE)
Wright-Patterson AFB, Ohio 45433

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

Comparison of Reliability Allocation Optimization Methods

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

AFIT Thesis

5. AUTHOR(S) (First name, middle initial, last name)

William Z. Spivey
2/Lt USAF

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1969	50	15

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	GRE/EE/69-2
c. N/A	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of the Dean, School of Engineering, Air Force Institute of Technology (AFIT-SE) Wright-Patterson Air Force Base, Ohio 45433.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

The reliability allocation problem has previously been solved by methods other than the classical optimization methods. This thesis brings together and compares three allocation models that are based on optimization methods. (U)

It was found that dynamic programming gives the optimal solution to the allocation problem. The model based on Lagrange multipliers gave a solution that was within 0.5% of the optimal solution given by dynamic programming. The linear programming algorithm gave a solution that was within 0.6% of the optimal solution. (U)

DD FORM 1 NOV 68 1473

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Lagrange multipliers						
Dynamic programming						
Reliability allocation						
Mathematical programming						